WARWICK
THE UNIVERSITY OF WARWICK

**Manuscript version: Author's Accepted Manuscript**
The version presented in WRAP is the author's accepted manuscript and may differ from the published version or Version of Record.

**Persistent WRAP URL:**
http://wrap.warwick.ac.uk/97737

**How to cite:**
Please refer to published version for the most recent bibliographic citation information. If a published version is known of, the repository item page linked to above, will contain details on accessing it.

**Copyright and reuse:**
The Warwick Research Archive Portal (WRAP) makes this work by researchers of the University of Warwick available open access under the following conditions.

Copyright © and all moral rights to the version of the paper presented here belong to the individual author(s) and/or other copyright owners. To the extent reasonable and practicable the material made available in WRAP has been checked for eligibility before being made available.

Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge. Provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way.

**Publisher's statement:**
Please refer to the repository item page, publisher's statement section, for further information.

For more information, please contact the WRAP Team at: wrap@warwick.ac.uk.

**warwick.ac.uk/lib-publications**

arXiv:1602.03755v1 [cs.DM] 11 Feb 2016

# Hitting Families of Schedules for Asynchronous Programs

Dmitry Chistikov, Rupak Majumdar, and Filip Niksic

Max Planck Institute for Software Systems (MPI-SWS)
Kaiserslautern and Saarbrücken, Germany
{dch,rupak,fniksic}@mpi-sws.org

**Abstract.** Asynchronous programming is a ubiquitous idiom for concurrent programming, where sequential units of code, called *events*, are scheduled and run atomically by a scheduler. While running, an event can post additional events for future execution by the scheduler. Asynchronous programs can have subtle bugs due to the non-deterministic scheduling of events, and a lot of recent research has focused on systematic testing of these programs.

Empirically, many bugs in asynchronous programs have small bug depth: that is, the number of events $d$ that must be scheduled in a specific order for a bug to be exposed is small. A natural question then is to find a $d$-hitting family of schedules: a set of schedules is a *$d$-hitting family* if for each set of $d$ events, and for each allowed ordering of these events, there is some schedule in the family that executes these events in this ordering. A $d$-hitting family is guaranteed to expose all bugs with $d$ events. By analyzing the structure of the tree of events in an asynchronous execution, we provide explicit constructions for small $d$-hitting families of schedules. When the tree is balanced, our construction is polylogarithmic in the number of events.

We have implemented our algorithm for computing $d$-hitting families on top of a race condition checker for web pages. We empirically confirm previous findings that many bugs occur with small bug depth. We demonstrate that even with $d = 2$ we are able to detect bugs in real web applications and that we get a small 3-hitting family for $d = 3$.

## 1 Introduction

Many bugs in concurrent programs depend on precise scheduling dependencies. Since the number of possible schedules is very large, such bugs can be very hard to find and reproduce. In practice, large concurrent systems are tested by running them under heavy load for days or weeks, with the hope that the rare schedules that expose bugs are chosen. In recent years, a number of testing approaches have been proposed that systematically execute all schedules of a given form. The main observation behind many of these techniques is that while concurrency bugs do depend on the precise ordering of several events, the *number* of events required to expose such bugs is small empirically [11,13]. This leads to

approaches such as context-bounded testing [12], delay-bounded testing [6], and PCT [4].

In this paper, we consider systematic testing of asynchronous programs. Asynchronous programming is a concurrent programming idiom where a single-threaded worker handles a sequence of *events*. Each event is executed atomically and can post other events into a task buffer for future execution. On completion of the currently executing event, a scheduler non-deterministically picks a new event from the task buffer and executes it on the worker. This programming style is common in many domains: from web programming using Ajax to servers and embedded systems.

The execution of an asynchronous program yields a partial order among the executed events, where an event $e_1$ happens before all events $e_2$ that it posts; this partial order is, in fact, a tree. A *schedule* is a linearization (a linear extension) of this partial order. Crucial to our testing is the notion of the *depth* of a bug: the minimum number of events that must be ordered in a specific way for the bug to be exposed by a schedule. For example, suppose there are two events $a$ and $b$ in the partial order of an execution. If a bug manifests itself only in schedules in which $a$ occurs before $b$, the bug depth is 2. If there are three events that must occur in a certain order for a bug to appear, the depth is 3, and so on. Race conditions (event $a$ writes, event $b$ reads, or vice versa) are of depth 2; atomicity violation bugs (event $a$ establishes an invariant, $b$ breaks it, $c$ assumes the invariant established by $a$) are of depth 3.

A schedule is said to *hit* a bug if it contains the events required to expose the bug in the required order. A natural question is to find a family of schedules that hits all potential bugs of depth $d$, for a fixed (and small) $d \geq 2$ —we call such a family a *$d$-hitting family* of schedules.

For $d = 2$, it turns out, surprisingly, that two schedules are enough, independent of the number of events in the partial order. These two schedules correspond to leftmost and rightmost DFS (depth-first) traversals of the tree, respectively. To the best of our knowledge, this observation was not used in systematic testing of asynchronous programs before! As we shall show, $d = 2$ is sufficient to expose a host of bugs in browser rendering of web pages.

For $d > 2$ and an execution tree of $n$ events, there is an obvious upper bound of $O(n^d)$ for the size of an optimal $d$-hitting family: for each choice of $d$ events, and each order of these events, take a schedule that hits this sequence. Indeed, an algorithm such as delay-bounded scheduling [6] with a delay bound of $d$ explores a $d$-hitting family of $O(n^d)$ schedules. Our main technical results show that this family can be exponentially sub-optimal. For $d = 3$ and a balanced tree on $n$ nodes, we show an explicit construction of a 3-hitting family of size $O(\log n)$, which is optimal up to a constant factor. For each $d > 3$, we show an explicit construction of a $d$-hitting family of size $f(d)(\log n)^{d-1}$, which is optimal up to a polynomial. Here $f(d)$ is an exponential function depending only on $d$. (Our construction for $d = 3$ works on a more general partial order that we call a *double tree*.) We also show a lower bound on the size of $d$-hitting families in terms of the height of the tree; in a dual way, for $n$ events without any happens-before

order between them, the size of any $d$-hitting family is at least $g(d) \log n$ for each $d > 2$.

For an asynchronous execution whose "height" (the maximum chain of events executed) is exponentially smaller than its "size" (the number of events), our construction gives an explicit test suite that is exponentially smaller than the size —in contrast to previous techniques for systematic testing.

We have implemented our algorithm on top of the R4 tool for testing race conditions in rendering web pages [9]. We empirically demonstrate that $d = 2$ is sufficient to expose many bugs in web pages (where we define a bug as a visual difference in the loaded pages). We show that our explicit construction of $d$-hitting families can be used to produce small test suites which can efficiently explore all bugs of depth $d$.

While we evaluate bug finding on web page loads, our techniques are general and can be applied to other classes of asynchronous programs as well.

Our notion of bug depth is similar to bug depth for shared-memory multi-threaded programs introduced in [4]. The notion in [4] is defined as the minimal number of additional constraints that guarantee an occurrence of the bug. Depending on the bug, this can be between half our notion and one less than our notion. However, unlike the asynchronous programs we consider, the partial order induced by an execution of a multi-threaded program can be arbitrary. Even for $d = 2$, finding an optimal $d$-hitting family for an arbitrary partial order is an NP-hard problem [20]; in fact, even approximating the optimal size is hard [8,5]. Burckhardt et al. [4] show an $O(mn^{d'-1})$ family for $m$ threads with $n$ instructions in total ($d'$ denotes bug depth according to their notion).

Our notion of $d$-hitting families is closely related to the notion of *order dimension* for a partial order, defined as the smallest number of linearizations, the intersection of which gives rise to the partial order [3,19,16]. Specifically, the size of an optimal 2-hitting family is the order dimension of a partial order, and the size of an optimal $d$-hitting family is a natural generalization. To the best of our knowledge, general $d$-hitting families have not been studied before for general partial orders. A version of the dimension ($d = 2$) called fractional dimension is known to be of use for approximation of some problems in scheduling theory [2]. Other generalizations of the dimension are also known (see, e.g., [18]), but, to the best of our knowledge, none of them is equivalent to ours.

Our contributions can be summarized as follows:

- We introduce $d$-hitting families as a common framework for systematic testing of asynchronous programs. We show that the size of optimal $d$-hitting families generalizes the order dimension for partial orders, and the families are natural combinatorial objects of independent interest.
- We provide explicit constructions for $d$-hitting families for trees that are close to optimal: up to a small constant factor for $d = 3$ and up to a polynomial for $d > 3$. In contrast to previous work, we provide families of schedules that can be exponentially smaller than the size of the partial order.

– We evaluate $d$-hitting families of schedules for practical examples and empirically demonstrate that for web page loading, even $d = 2$ is effective in identifying bugs.

## 2 Hitting families of schedules

In this section, we first recall the standard terminology of partial orders, and then proceed to define schedules (linearizations of these partial orders) and hitting families of schedules.

**Preliminaries: Partial orders.** A *partial order* (also known as a partially ordered set, or a poset) is a pair $(\mathcal{P}, \leq)$ where $\mathcal{P}$ is a set and $\leq$ is a binary relation on $\mathcal{P}$ that is:

1) reflexive: $x \leq x$ for all $x \in \mathcal{P}$,
2) antisymmetric: $x \leq y$ and $y \leq x$ imply $x = y$ for all $x, y \in \mathcal{P}$,
3) transitive: $x \leq y$ and $y \leq z$ imply $x \leq z$ for all $x, y, z \in \mathcal{P}$.

One typically uses $\mathcal{P}$ to refer to $(\mathcal{P}, \leq)$. We will refer to elements of partial orders as *events*; the *size* of $\mathcal{P}$ is the number of events in it, $|\mathcal{P}|$.

The relation $x \leq y$ is also written as $x \leq_{\mathcal{P}} y$ and as $y \geq x$; the event $x$ is a *predecessor* of $y$, and $y$ is a *successor* of $x$. One writes $x < y$ iff $x \leq y$ and $x \neq y$. Furthermore, $x$ is an *immediate predecessor* of $y$ (and $y$ is an *immediate successor* of $x$) if $x < y$ but there is no $z \in \mathcal{P}$ such that $x < z < y$. The *Hasse diagram* of a partial order $\mathcal{P}$ is a directed graph where the set of vertices is $\mathcal{P}$ and an edge $(x, y)$ exists if and only if $x$ is an immediate predecessor of $y$. Partial orders are sometimes identified with their Hasse diagrams.

Events $x$ and $y$ are *comparable* iff $x \leq y$ or $y \leq x$. Otherwise they are *incomparable*, which is written as $x \mathrel{||} y$. Partial orders $(\mathcal{P}_1, \leq_1)$ and $(\mathcal{P}_2, \leq_2)$ are *disjoint* if $\mathcal{P}_1 \cap \mathcal{P}_2 = \emptyset$; the *parallel composition* (or *disjoint union*) of such partial orders is the partial order $(\mathcal{P}, \leq)$ where $\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2$ and $x \leq y$ iff $x, y \in \mathcal{P}_k$ for some $k \in \{1, 2\}$ and $x \leq_k y$. In this partial order, which we will denote by $\mathcal{P}_1 \parallel \mathcal{P}_2$, any two events not coming from a single $\mathcal{P}_k$ are incomparable: $x_1 \in \mathcal{P}_1$ and $x_2 \in \mathcal{P}_2$ imply $x_1 \mathrel{||} x_2$.

For a partial order $(\mathcal{P}, \leq)$ and a subset $\mathcal{Q} \subseteq \mathcal{P}$, the *restriction* of $(\mathcal{P}, \leq)$ to $\mathcal{Q}$ is the partial order $(\mathcal{Q}, \leq_{\mathcal{Q}})$ in which, for all $x, y \in \mathcal{Q}$, $x \leq_{\mathcal{Q}} y$ if and only if $x \leq y$. Instead of $\leq_{\mathcal{Q}}$ one usually writes $\leq$, thus denoting the restriction by $(\mathcal{Q}, \leq)$. We will also say that the partial order $\mathcal{P}$ *contains* the partial order $\mathcal{Q}$. In general, partial orders $(\mathcal{P}_1, \leq_1)$ and $(\mathcal{P}_2, \leq_2)$ are *isomorphic* iff there exists an isomorphism $f \colon \mathcal{P}_1 \to \mathcal{P}_2$: a bijective mapping that respects the ordering, i.e., with $x \leq_1 y$ iff $f(x) \leq_2 f(y)$ for all $x, y \in \mathcal{P}_1$. Containment of partial orders is usually understood up to isomorphism.

**Schedules and their families.** A partial order is *linear* (or total) if all its events are pairwise comparable. A linearization (linear extension) of the partial order $(\mathcal{P}, \leq)$ is a partial order of the form $(\mathcal{P}, \leq')$ that is linear. We call linearizations (linear extensions) of $\mathcal{P}$ *schedules*. In other words, a schedule $\alpha$ is a

permutation of the elements of $\mathcal{P}$ that *respects* $\mathcal{P}$, i.e., *respects* all constraints of the form $x \leq y$ from $\mathcal{P}$: for all pairs $x, y \in \mathcal{P}$, whenever $x \leq_{\mathcal{P}} y$, it also holds that $x \leq_\alpha y$. We denote the set of all possible schedules by $S(\mathcal{P})$; a *family* of schedules for $\mathcal{P}$ is simply a subset of $S(\mathcal{P})$.

In what follows, we often treat schedules as words and families of schedules as languages. Indeed, let $\mathcal{P}$ have $n$ elements $\{v_1, \ldots, v_n\}$, then any schedule $\alpha$ can be viewed as a word of length $n$ over the alphabet $\{v_1, \ldots, v_n\}$ where each letter occurs exactly once. We say that $\alpha$ *schedules* events in the order of occurrences of letters in the word that represents it.

Suppose $\alpha_1$ and $\alpha_2$ are schedules for disjoint partial orders $\mathcal{P}_1$ and $\mathcal{P}_2$; then $\alpha_1 \cdot \alpha_2$ is a schedule for the partial order $\mathcal{P}_1 \parallel \mathcal{P}_2$ that first schedules all events from $\mathcal{P}_1$ according to $\alpha_1$ and then all events from $\mathcal{P}_2$ according to $\alpha_2$. Note that we will use the $\cdot$ to concatenate schedules (as well as individual events); since some of our partially ordered sets will contain strings, we will use the full stop . to denote concatenation "inside" an event.

**Admissible tuples and *d*-hitting families.** Fix a partial order $\mathcal{P}$ and let $\boldsymbol{a} = (a_1, \ldots, a_d)$ be a tuple of $d \geq 2$ distinct elements of $\mathcal{P}$; we call such tuples *d-tuples*. Suppose $\alpha$ is a schedule for $\mathcal{P}$; then the schedule $\alpha$ *hits* the tuple $\boldsymbol{a}$ if the restriction of $\alpha$ to the set $\{a_1, \ldots, a_d\}$ is the sequence $a_1 \cdot \ldots \cdot a_d$.

Note that for a tuple $\boldsymbol{a}$ to have a schedule that hits $\boldsymbol{a}$ it is necessary and sufficient that $\boldsymbol{a}$ respect $\mathcal{P}$; this condition is equivalent to the condition that $a_i \leq a_j$ or $a_i \shuffle a_j$ whenever $1 \leq i \leq j \leq d$. We call $d$-tuples satisfying this condition *admissible*.

**Definition 1 ($d$-hitting family).** A family of schedules $F$ for $\mathcal{P}$ is *d-hitting* if for every admissible $d$-tuple $\boldsymbol{a}$ there is a schedule $\alpha \in F$ that hits $\boldsymbol{a}$.

It is straightforward that every $\mathcal{P}$ with $|\mathcal{P}| = n$ has a $d$-hitting family of size at most $\binom{n}{d} \cdot d! \leq n^d$: just take any hitting schedule for each admissible $d$-tuple, of which there are at most $\binom{n}{d} \cdot d!$. For $d = 2$, the size of the smallest 2-hitting family is known as the dimension of the partial order [3,19]. Computing and even approximating the dimension for general partial orders is known to be a hard problem [20,8,5]. In the remainder of the paper, we focus on $d$-hitting families for specific partial orders, most importantly trees, (which represent happens-before relations of asynchronous programs). We first consider two simple examples.

**Example 2 (chain).** Consider a chain of $n$ events (a linear order): $\mathcal{C}_n = \{1, \ldots, n\}$ with $1 < 2 < \ldots < n$. This partial order has a unique schedule: $\alpha = 1 \cdot 2 \cdot \ldots \cdot n$; a $d$-tuple $\boldsymbol{a} = (a_1, \ldots, a_d)$ is admissible iff $a_1 < \ldots < a_n$, and $\alpha$ hits all such $d$-tuples. Thus, for any $d$, the family $F = \{\alpha\}$ is a $d$-hitting family for $\mathcal{C}_n$.

**Example 3 (chain with independent event).** Consider $\mathcal{C}_n \parallel \{\dagger\}$, the disjoint union of $\mathcal{C}_n$ from Example 2 and a singleton $\{\dagger\}$. There are $n + 1$ possible schedules, depending on how $\dagger$ is positioned with respect to the chain: $\alpha_0 = \dagger \cdot 1 \cdot 2 \cdot \ldots \cdot n$, $\alpha_1 = 1 \cdot \dagger \cdot 2 \cdot \ldots \cdot n$, …, $\alpha_n = 1 \cdot 2 \cdot \ldots \cdot n \cdot \dagger$. For $d = 2$, admissible pairs are of the form $(i, j)$ with $i < j$, $(\dagger, i)$, and $(i, \dagger)$ for all $1 \leq i \leq n$; the family $F_2 = \{\alpha_0, \alpha_n\}$ is the smallest 2-hitting family. Now consider $d = 3$.

Note that all triples $(i, \dagger, i+1)$ with $1 \leq i \leq n-1$, as well as $(\dagger, 1, 2)$ and $(n-1, n, \dagger)$, are admissible, and each of them is hit by a unique schedule. Therefore, the smallest 3-hitting family of schedules consists of all $n+1$ schedules: $F_3 = \{\alpha_0, \ldots, \alpha_n\}$. For $d \geq 4$, it remains to observe that every $d$-hitting family is necessarily $d'$-hitting for $2 \leq d' \leq d$, hence $F_3$ is optimal for all $d \geq 3$.

An important **corollary** of this example is that, for any $d \geq 3$ and any partial order $\mathcal{P}$ non-isomorphic to $\mathcal{C}_n$ with $n = |\mathcal{P}|$, every $d$-hitting family must contain at least $h(\mathcal{P})+1$ schedules, where $h(\mathcal{P})$ denotes the *height* of the partial order $\mathcal{P}$ (sometimes called *length*): the maximal cardinality of a chain (a set of pairwise comparable events) in $\mathcal{P}$.

## 3 Hitting families of schedules for trees

### 3.1 Definitions and overview

Consider a complete binary tree of height $h$ with edges directed from the root. This tree is the Hasse diagram of a partial order $\mathcal{T}^h$, unique up to isomorphism; we will apply tree terminology to $\mathcal{T}^h$ itself. The root of $\mathcal{T}^h$ forms the 0th *layer*, its children the 1st layer and so on. The maximum $k$ such that $\mathcal{T}^h$ has an element in the $k$th layer is height of the tree $\mathcal{T}^h$. We will assume that elements of $\mathcal{T}^h$ are strings: $\mathcal{T}^h = \{0,1\}^{\leq h}$ with $x \leq y$ for $x, y \in \mathcal{T}^h$ iff $x$ is a prefix of $y$. The $k$th layer of $\mathcal{T}^h$ is $\{0,1\}^k$, and nodes of the $h$th layer are *leaves*. Unless $x \in \mathcal{T}^h$ is leaf, nodes $x.0$ and $x.1$ are left- and right-children of $x$, respectively. (Recall that the full stop denotes concatenation of strings, with the purpose of distinguishing individual strings and their sequences.) The tree $\mathcal{T}^h$ has $n = 2^{h+1} - 1$ nodes.

The central question that we study in this paper is as follows: How big are optimal $d$-hitting families of schedules for $\mathcal{T}^h$ with $n$ nodes?

As it turns out, for $\mathcal{T}^h$ very efficient constructions of $d$-hitting families exist. It is, in fact, possible, to find such families that have size *exponentially smaller* than $n$, the number of events. More specifically, we prove the following results ($h$ is the height of the partial order—the size of the longest chain):

1. For arbitrary $d \geq 3$, there is a simple $d$-hitting family of size $O(n^{d-2})$ (Claim 5 in the following subsection 3.2).
2. For $d = 3$, there is a 3-hitting family of size $O(h)$ (Theorem 7 in Section 4).
3. For arbitrary $d \geq 3$, there is a $d$-hitting family of size $O(h^{d-1})$ (Theorem 10 in Section 5).

Our main technical results are Theorems 7 and 10, shown in the next sections—where they are stated for complete binary trees, with $h = \log(n+1) - 1$. (Arbitrary trees are, of course, contained in these complete trees.) The remainder of this section is structured as follows. In subsection 3.2, we prove, as a warm-up, Claim 5. After this, in subsection 3.3, we show that the problem of finding families of schedules with size smaller than $n$ turns out to be tricky even when there are no dependencies between events at all. This problem arises as a sub-problem when considering trees (as, indeed, there are no dependencies between the leaves in a tree), and thus our main constructions in Sections 4 and 5 must be at least as agile.

### 3.2 Warm-up: $d$-hitting families of size $O(n^{d-2})$

**Claim 4.** The smallest 2-hitting family of schedules for $\mathcal{T}^h$ has size 2.

The construction is as follows. Take $F_{\mathsf{dfs}} = \{\lambda, \rho\}$ where $\lambda$ and $\rho$ are left-to-right and right-to-left DFS (depth-first) traversals of $\mathcal{T}^h$, respectively. More formally, these schedules are defined as follows: for $x, y \in \mathcal{T}^h$, $x \leq_\lambda y$ if either $x \leq y$ (i.e., $x$ is a prefix of $y$) or $x = u.0.x'$ and $y = u.1.y'$ for some strings $u, x', y' \in \{0,1\}^*$; $x \leq_\rho y$ if either $x \leq y$ or $x = u.1.x'$ and $y = u.0.y'$. For instance, $\mathcal{T}^2$ has $\lambda = \varepsilon \cdot 0 \cdot 00 \cdot 01 \cdot 1 \cdot 10 \cdot 11$ and $\rho = \varepsilon \cdot 1 \cdot 11 \cdot 10 \cdot 0 \cdot 01 \cdot 00$. The family $F_{\mathsf{dfs}}$ is 2-hitting: all admissible pairs $(x, y)$ satisfy either $x \leq y$, in which case they are hit by any possible schedule, or $x \shortparallel y$, in which case neither is a prefix of the other, $x = u.a.x'$ and $y = u.\bar{a}.y'$ with $\{a, \bar{a}\} = \{0, 1\}$, so $\lambda$ and $\rho$ schedule them in reverse orders. Since it is clear that a family of size 1 cannot be 2-hitting for $\mathcal{T}^h$ with $h \geq 1$ (as $\mathcal{T}^h$ contains at least one pair of incomparable elements), the family $F_{\mathsf{dfs}}$ is optimal.

Based on this construction for $d = 2$, it is possible to find $d$-hitting families for $d \geq 3$ that have size $o(n^d)$ where $n = 2^{h+1} - 1$ is the number of events in $\mathcal{T}^h$:

**Claim 5.** For any $d \geq 3$, $\mathcal{T}^h$ has a $d$-hitting family of schedules of size $O(n^{d-2})$.

Indeed, group all admissible $d$-tuples $\boldsymbol{a} = (a_1, \ldots, a_d)$ into bags agreeing on $a_1, \ldots, a_{d-2}$. For each bag, construct a pair of schedules $\lambda' = \lambda'(a_1, \ldots, a_{d-2})$ and $\rho' = \rho'(a_1, \ldots, a_{d-2})$ as follows. In both $\lambda'$ and $\rho'$, first *schedule* $a_1, \ldots, a_{d-2}$: that is, start with an empty sequence of events, iterate over $k = 1, \ldots, d-2$, and, for each $k$, append to the sequence all events $x \in \mathcal{T}^h$ such that $x \leq a_k$. The order in which these $x$es are appended is chosen in the unique way that respects the partial order $\mathcal{T}^h$. Events that are predecessors of several $a_k$ are only scheduled once, for the least $k$. Note that no $a_k$, $1 \leq k \leq d$, is a predecessor of any $a_j$ for $j < k$, because otherwise the $d$-tuple $\boldsymbol{a} = (a_1, \ldots, a_d)$ is not admissible. After this, the events of $\mathcal{T}^h$ that have not been scheduled yet form a disjoint union of several binary trees. The schedule $\lambda'$ then schedules all events according to how the left-to-right DFS traversal $\lambda$ would work on $\mathcal{T}^h$, omitting all events that have already been scheduled, and the schedule $\rho'$ does the same based on $\rho$. As a result, these two schedules hit all admissible $d$-tuples that agree on $a_1, \ldots, a_{d-2}$; collecting all such schedules for all possible $a_1, \ldots, a_{d-2}$ makes a $d$-hitting family for $\mathcal{T}^h$ of size at most $2n^{d-2}$.

### 3.3 Antichains: $d$-hitting families of size $f(d) \log n$

An *antichain* in a partial order $\mathcal{P}$ is a subset of $\mathcal{P}$ in which every two elements are incomparable. For example, a complete binary tree with $m$ nodes contains an antichain of size $\lfloor m/2 \rfloor$: the set of its leaves. Thus, any $d$-hitting family of sublinear size for the tree must necessarily extend a $d$-hitting family of sublinear size for the antichain—a problem of independent interest that we study in this section. We regard the antichain as a standalone partial order, $\mathcal{A}_n = \{v_1\} \parallel \{v_2\} \parallel \ldots \parallel \{v_n\}$. The set of all schedules for $\mathcal{A}_n$ is $S_n$, the set of all permutations,

and the set of all admissible $d$-tuples is the set of all $d$-arrangements of these $n$ events.

**Theorem 6.** *For any $d \geq 3$, the smallest $d$-hitting family for $\mathcal{A}_n$ has size between $g(d)\log n - O(1)$ and $f(d)\log n$, where $g(d) \geq d/2\log(d+1)$ and $f(d) \leq d!\,d$.*

We sketch the proof of Theorem 6 in the remainder of this section. We will show how to obtain the upper bound by two different means: with the probabilistic method and with a greedy approach. From the results of the following section 4 one can extract a derandomization for $d = 3$, also with size $O(\log n)$; and section 5 achieves size $f(d)(\log n)^{d-1}$ for $d \geq 3$. In the current section we also show a lower bound based on a counting argument.

**Upper bound: Probabilistic method.** Consider a family of schedules $F = \{\alpha_1, \ldots, \alpha_k\}$ where each $\alpha_i$ is chosen independently and uniformly at random from $S_n$; the parameter $k$ will be chosen later. Fix any admissible $\boldsymbol{a} = (a_1, \ldots, a_d)$. What is the probability that a specific $\alpha_i$ does not hit $\boldsymbol{a}$? A random permutation arranges $a_1, \ldots, a_d$ in one of $d!$ possible orders without preference to any of them, so this probability is $1 - 1/d!$. Since all $\alpha_i$ are chosen independently, the probability that none of them hits $\boldsymbol{a}$ is $(1-1/d!)^k$. By the union bound, the probability that *at least one* $d$-tuple $\boldsymbol{a}$ is not hit by any of $\alpha_i$ does not exceed $p = n^d \cdot (1-1/d!)^k$.

Now observe that this value of $p$ is exactly the probability that $F$ is not a $d$-hitting family. If we now choose $k$ in such a way that $p < 1$, then the probability of $F$ being a $d$-hitting family is non-zero, i.e., a $d$-hitting family of size $k$ exists. Calculation shows that $k > d!\,d \cdot \ln n$ suffices.

The probabilistic method, a classic tool in combinatorics, is due to Erdős [1].

**Upper bound: Greedy approach.** We exploit the following connection between $d$-hitting families and *set covers*. Recall that in a set cover problem one is given a number of sets, $R_1, \ldots, R_s$, and the goal is find a small number of these sets whose union is equal to $R = R_1 \cup \ldots \cup R_s$. A set $R_i$ covers an element $e \in R$ iff $e \in R_i$, and this covering is essentially the same as hitting in $d$-hitting families: elements $e \in R$ are admissible $d$-tuples $\boldsymbol{a} = (a_1, \ldots, a_d)$, and each schedule $\alpha$ corresponds to a set $R_\alpha$ that contains all $d$-tuples $\boldsymbol{a}$ that it hits. A $d$-hitting family of schedules is then the same as a set cover.

A well-known approach to the set cover problem is the greedy algorithm, which in our setting works as follows. Initialize a list of all admissible $\boldsymbol{a} = (a_1, \ldots, a_d)$; on each step, pick some schedule $\alpha$ that hits the largest number of tuples in the list, and cross out all these tuples. Terminate when the list is empty; the set of all picked schedules is a $d$-hitting family.

While this algorithm can be used for any partial order $\mathcal{P}$, in our case we can estimate the quality of its output. The so-called greedy covering lemma by Sapozhenko [15] or a more widely known Lovász-Stein theorem [10,17] gives an explicit upper bound on the size of the obtained greedy cover in terms of $|R|$ and the density of the instance (the smallest $\gamma$ such that every $e \in R$ belongs to at least $\gamma s$ out of $s$ sets). In our case, $|R| \leq n^d$, and the density is $1/d!$; the obtained upper bound on the size of the smallest $d$-hitting family is $d!\,d \cdot \ln n - \Theta(d!\,d\log d)$.

**Lower bound.** Consider the case $d = 3$. Take any 3-hitting family $F = \{\alpha_1, \ldots, \alpha_k\}$ and consider the binary matrix $B = (b_{ij})$ of size $k \times (n-1)$ where $b_{ij} = 1$ iff the schedule $\alpha_i$ places event $v_j$ before $v_n$. We claim that all columns of $B$ are pairwise distinct. Indeed, if for some $j' \neq j''$ and all $i$ it holds that $b_{ij'} = b_{ij''}$, then no schedule from $F$ can place $v_{j'}$ before $v_n$ without also placing $v_{j''}$ before $v_n$, and vice versa. This means that no schedule from $F$ hits the 3-tuples $\boldsymbol{a}' = (v_{j'}, v_n, v_{j''})$ and $\boldsymbol{a}'' = (v_{j''}, v_n, v_{j'})$, so $F$ cannot be 3-hitting.

Since all columns of $B$ are pairwise distinct and $B$ is a 0/1-matrix, it follows that the number of columns, $n-1$, cannot be greater than the number of all subsets of its rows, $2^k$. From $n - 1 \leq 2^k$ we deduce that $k \geq \log(n-1)$. The construction in the general case $d \geq 3$ is analogous.

## 4 3-hitting families of size $O(\log n)$

In this section we construct explicit 3-hitting families of schedules of logarithmic size for partial orders that we call double trees. Double trees are extensions of trees, so restriction of these 3-hitting families to appropriate subsets gives explicit 3-hitting families for trees and for antichains, also of logarithmic size.

The *(binary) double tree* of half-height $h \geq 1$ is the partial order $\mathcal{D}$ defined as follows. Intuitively, each $\mathcal{D}^h$ is a parallel composition (disjoint union) of two copies of $\mathcal{D}^{h-1}$, with additional top and bottom (largest and smallest) events; and the induction basis is that $\mathcal{D}^0$ consists of a single event.

More precisely, (the Hasse diagram of) $\mathcal{D}$ consists of two complete binary trees of height $h$ that share their set of $2^h$ leaves; in the first tree, the edges are directed from the root to the leaves, and in the second tree, from the leaves to the root. Formally, $\mathcal{D}^h = \{-1, +1\} \times \{0,1\}^{\leq h-1} \cup \{0\} \times \{0,1\}^h$; note that the cardinality of this set is $3 \cdot 2^h - 2$. Each event $x = (s_x, x') \in \mathcal{D}^h$ belongs to either of trees ($s_x \in \{-1, +1\}$) or is a shared leaf ($s_x = 0$). We define the ordering by taking the transitive closure of the following relation: let $x = (s_x, x')$ and $y = (s_y, y')$ be events of $\mathcal{D}^h$; if $\{s_x, s_y\} \subseteq \{-1, 0\}$, then $x \leq y$ whenever $x'$ is a prefix of $y'$; and if $\{s_x, s_y\} \subseteq \{0, +1\}$, then $x \leq y$ whenever $y'$ is a prefix of $x'$. (Note that all events $x, y$ with $s_x = s_y = 0$ are pairwise incomparable.)

**Theorem 7.** *The smallest 3-hitting family for the double tree $\mathcal{D}^h$ with $n = 3 \cdot 2^h - 2$ events has size between $2h + 1 = 2\log n - O(1)$ and $4h = 4\log n - O(1)$.*

Recall that a double tree with $3 \cdot 2^h - 2$ events contains a complete binary tree with $2 \cdot 2^h - 1$ nodes, which in turn contains an antichain of size $2^h$. As a corollary, $\mathcal{T}^h$, a tree with $n = 2 \cdot 2^h - 1$ nodes has a 3-hitting family of size $4h = 4\log(n+1) - 4$. Similarly, $\mathcal{A}_n$, an antichain of size $n$, has a 3-hitting family of size $4 \log n$. Unlike the constructions from subsection 3.3, the construction of Theorem 7 is explicit.

**Corollary 8.** *The smallest 3-hitting family for an arbitrary (not necessarily balanced) tree of height $h$ and outdegree at most $\Delta$ has size between $h + 1$ and $4h \log \Delta$, unless the tree is a single chain (Example 2).*

Note that lower bounds proportional to $h$ follow from Example 3. We describe the construction of Theorem 7 below.

**Matrix notation.** We use the following notation for families of schedules. Let $\mathcal{P}$ be a partial order, $|\mathcal{P}| = n$. Let $F$ be a family of schedules for $\mathcal{P}$, $|F| = m$. We then write

$$F = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

where $F = \{\alpha_1, \dots, \alpha_m\}$ and $\alpha_i = a_{i1} \cdot a_{i2} \cdot \ldots \cdot a_{in}$ for $1 \le i \le m$. In other words, a family of $m$ schedules for an $n$-sized partial order is written as an $m \times n$-matrix whose entries are elements of $\mathcal{P}$, with no element appearing more than once in any row. In particular, if $\alpha$ is a schedule for $\mathcal{P}$, then we represent it with a row vector. The union of families naturally corresponds to stacking of matrices: $F_1 \cup F_2 = \begin{pmatrix} F_1 \\ F_2 \end{pmatrix}$, and putting two matrices of the same height $m$ next to each other corresponds to concatenating two families of size $m$, in order to obtain a family of size $m$ for the union of two partial orders: $\begin{pmatrix} F_1 & F_2 \end{pmatrix}$.

**Construction of 3-hitting families for double trees.** We define the families of schedules using induction on $h$; in matrix notation, the families will be denoted and structured as follows:

$$M_h = \begin{bmatrix} A_h & B_h \\ C_h & D_h \end{bmatrix}$$

where all four blocks are of size $(3 \cdot 2^{h-1} - 1) \times 2h$; in total, $M^h$ will contain $4h$ schedules, each with $3 \cdot 2^h - 2$ events.

Base case, $h = 1$:

$$\left[\, A_1 \middle| B_1 \,\right] = \left[\, C_1 \middle| D_1 \,\right] = \left[\begin{array}{cc|cc} (-1,\varepsilon) & (0,0) & (0,1) & (+1,\varepsilon) \\ (-1,\varepsilon) & (0,1) & (0,0) & (+1,\varepsilon) \end{array}\right].$$

Note that $M_1$ is redundant: it specifies both possible schedules two times. However, this redundancy disappears in the inductive step.

Inductive step from $h \ge 1$ to $h + 1$: Note that, for $\ell \in \{0, 1\}$, restricting $\mathcal{D}^{h+1}$ to events of the form $(s, x')$ where $x' = \ell.x''$ leads to a partial order isomorphic to $\mathcal{D}^h$; these two partial orders are disjoint, and we denote them by $\mathcal{D}^h(\ell)$, $\ell \in \{0, 1\}$; in fact, $\mathcal{D}^h(0) \cup \mathcal{D}^h(1) \cup \{(-1, \varepsilon), (+1, \varepsilon)\}$ forms a partition of $\mathcal{D}^{h+1}$. We assume that the matrix $M_h$ is known (the inductive hypothesis); for $\ell \in \{0, 1\}$, we denote its image under the (entry-wise) mapping $(s, x') \mapsto (s, \ell.x')$ by $M_h(\ell)$. In other words, $M_h(\ell)$ is the matrix that defines our (soon proved to be 3-hitting) family of schedules for $\mathcal{D}^h(\ell)$; we will also apply the same notation to $A$, $B$, $C$, and $D$.

Finally, we will need two auxiliary schedules for double trees, which we call *left* and *right* traversals. The left traversal $\lambda$ of $\mathcal{D}^{h+1}$ is defined inductively as follows: it first schedules $(-1, \varepsilon)$, then takes the left traversal of $\mathcal{D}^h(0)$, then the left traversal of $\mathcal{D}^h(1)$, and then schedules $(+1, \varepsilon)$. The right traversal $\rho$ is

defined symmetrically. Denote by $\lambda(\ell)$ and $\rho(\ell)$ left and right traversals of $\mathcal{D}^h(\ell)$, respectively (we omit reference to $h$ since this does not create confusion). Then

$$A_{h+1} = \left[\begin{array}{c|c|c} (-1,\varepsilon) & & \\ \vdots & A_h(0) & A_h(1) \\ (-1,\varepsilon) & & \\ \hline (-1,\varepsilon) & \lambda(0) & \\ \hline (-1,\varepsilon) & \lambda(1) & \end{array}\right], \quad B_{h+1} = \left[\begin{array}{c|c|c} & & (+1,\varepsilon) \\ B_h(1) & B_h(0) & \vdots \\ & & (+1,\varepsilon) \\ \hline \lambda(1) & & (+1,\varepsilon) \\ \hline \lambda(0) & & (+1,\varepsilon) \end{array}\right],$$

$$C_{h+1} = \left[\begin{array}{c|c|c} (-1,\varepsilon) & & \\ \vdots & C_h(1) & C_h(0) \\ (-1,\varepsilon) & & \\ \hline (-1,\varepsilon) & \rho(0) & \\ \hline (-1,\varepsilon) & \rho(1) & \end{array}\right], \quad D_{h+1} = \left[\begin{array}{c|c|c} & & (+1,\varepsilon) \\ D_h(0) & D_h(1) & \vdots \\ & & (+1,\varepsilon) \\ \hline \rho(1) & & (+1,\varepsilon) \\ \hline \rho(0) & & (+1,\varepsilon) \end{array}\right].$$

Our result is that, for each $h$, $M_h$ is a 3-hitting family of schedules for $\mathcal{D}^h$. The key part of the proof relies on the following auxiliary property, which is a stronger form of the 2-hitting condition.

**Lemma 9.** *For any pair of distinct events $\boldsymbol{a} = (a_1, a_2)$ from $\mathcal{D}^h$, if there is a schedule for $\mathcal{D}^h$ that hits $\boldsymbol{a}$, then each of the matrices $\left[A_h \middle| B_h\right]$ and $\left[C_h \middle| D_h\right]$ contains a schedule for $\mathcal{D}^h$ where $a_1$ is placed in the first half and $a_2$ is placed in the second half.*

## 5 $d$-hitting families for $d \geq 3$ of size $f(d)(\log n)^{d-1}$

Fix some $d$ and let $\mathcal{T}^h$ be a complete binary tree of height $h$, as defined in subsection 3.1. In this section we prove the following theorem.

**Theorem 10.** *For any $d \geq 2$ the complete binary tree of height $h$ has a $d$-hitting family of schedules of size $\exp(d) \cdot h^{d-1}$.*

Note that in terms of the number of nodes of $\mathcal{T}^h$, which is $n = 2^{h+1} - 1$, Theorem 10 gives a $d$-hitting family of size polylogarithmic in $n$. The proof of the theorem is constructive, and we divide it into three steps. The precise meaning to the steps relies on auxiliary notions of a *pattern* and of $d$-tuples *conforming to* a pattern; we give all necessary definitions below.

**Lemma 11.** *For each admissible $d$-tuple $\boldsymbol{a} = (a_1, \ldots, a_d)$ there exists a pattern $p$ such that $\boldsymbol{a}$ conforms to $p$.*

**Lemma 12.** *For each pattern $p$ there exists a schedule $\alpha_p$ that hits all $d$-tuples $\boldsymbol{a}$ that conform to $p$.*

**Lemma 13.** *The total number of patterns, up to isomorphism, does not exceed $\exp(d) \cdot h^{d-1}$.*

The statement of Theorem 10 follows easily from these lemmas. The key insight is the definition of the pattern and the construction of Lemma 12.

In the sequel, for partial orders that are trees directed from the root we will use the standard terminology for graphs and trees (relying on Hasse diagrams): node, outdegree, siblings, 0- and 1-principal subtree of a node, isomorphism. We denote the parent of a node $u$ by $\operatorname{par} u$ and the *least common ancestor* of nodes $u$ and $v$ by $\operatorname{lca}(u, v)$.

If $T$ is a tree and $X \subseteq T$ is a subset of its nodes, then by $[X]$ we denote the lca-closure of $X$: the smallest set $Y \subseteq T$ such that, first, $X \subseteq Y$ and, second, for any $y_1, y_2 \in Y$ it holds that $\operatorname{lca}(y_1, y_2) \in Y$. The following claim is a variation of a folklore Lemma 1 in [7].

**Claim 14.** $|[X]| \leq 2\,|X| - 1$.

**Definition 15 (pattern).** A *pattern* is a quintuple $p = (D, \preccurlyeq, s, \ell, \pi)$ where:

— $d \leq |D| \leq 2d - 1$,
— $(D, \preccurlyeq)$ is a partial order which is, moreover, a tree directed from the root,
— the number of non-leaf nodes in $(D, \preccurlyeq)$ does not exceed $d - 1$,
— each node of $(D, \preccurlyeq)$ has outdegree at most 2,
— the partial function $s\colon D \rightharpoonup \{0, 1\}$ specifies, for each pair of siblings $v_1, v_2$ in $(D, \preccurlyeq)$, which is the left and which is the right child of its parent: $s(v_t) = 0$ and $s(v_{3-t}) = 1$ for some $t \in \{1, 2\}$; the value of $s$ is undefined on all other nodes of $D$,
— the partial function $\ell\colon D \rightharpoonup \{0, 1, \ldots, h - 1\}$ associates a *layer* with each non-leaf node of $(D, \preccurlyeq)$, so that $u \prec v$ implies $\ell(u) < \ell(v)$; the value of $\ell$ is undefined on all leaves of $D$, and
— $\pi$ is a schedule for $(D, \preccurlyeq)$.

We remind the reader that the symbol $\leq$ refers to the same partial order as $\mathcal{T}^h$.

**Definition 16 (conformance).** Take any pattern $p = (D, \preccurlyeq, s, \ell, \pi)$ and any tuple $\boldsymbol{a} = (a_1, \ldots, a_d)$ of $d$ distinct elements of the partial order $\mathcal{T}^h$. Consider the set $\{a_1, \ldots, a_d\}$: the restriction of $\leq$ to its lca-closure $A = [\{a_1, \ldots, a_d\}]$ is a binary tree, $(A, \leq)$. Suppose that the following conditions are satisfied:

a) the trees $(D, \preccurlyeq)$ and $(A, \leq)$ are isomorphic: there exists a bijective mapping $i\colon D \to A$ such that $v_1 \preccurlyeq v_2$ in $D$ iff $i(v_1) \leq i(v_2)$ in $\mathcal{T}^h$;
b) the partial function $s$ correctly indicates left- and right-subtree relations: for any $v \in D$, $s(v) = b \in \{0, 1\}$ if and only if $i(v)$ lies in the $b$-principal subtree of $i(\operatorname{par}(v))$;
c) the partial function $\ell$ correctly specifies the layer inside $\mathcal{T}^h$: for any non-leaf $v \in D$, $\ell(v) = |i(v)|$; recall that elements of $\mathcal{T}^h$ are binary strings from $\{0, 1\}^{\leq h}$;
d) the schedule $\pi$ for $(D, \preccurlyeq)$ hits the tuple $i^{-1}(\boldsymbol{a}) = (i^{-1}(a_1), \ldots, i^{-1}(a_d))$.

Then we shall say that the tuple $\boldsymbol{a}$ *conforms to* the pattern $p$.

We now sketch the proof of Lemma 12. Fix any pattern $p = (D, \preccurlyeq, s, \ell, \pi)$. Recall that we need to find a schedule $\alpha_p$ that hits all $d$-tuples $\boldsymbol{a} = (a_1, \ldots, a_d)$ conforming to $p$. We will pursue the following strategy. We will cut the tree $\mathcal{T}^h$ into multiple pieces; this cutting will be entirely determined by the pattern $p$, independent of any individual $\boldsymbol{a}$. Each piece in the cutting will be associated with some element $c \in D$, so that each element of $D$ can have several pieces associated with it. In fact, every piece will form a subtree of $\mathcal{T}^h$ (although this will be of little importance). The key property is that, for every $d$-tuple $\boldsymbol{a} = (a_1, \ldots, a_d)$ conforming to $p$, if $i$ is the isomorphism from Definition 16, then each event $a_k$, $1 \leq k \leq d$, will belong to a piece associated with $i^{-1}(a_k)$. As a result, the desired schedule $\alpha_p$ can be obtained in the following way: arrange the pieces according to how $\pi$ schedules elements of $D$ and pick any possible schedule inside each piece. This schedule will be guaranteed to meet the requirements of the lemma.

## 6 Experiments

We have applied the theoretical results presented in this paper to systematic testing of web pages. Web pages are event-driven: As the browser parses the page, it concurrently executes JavaScript code registered to handle various automatic or user-triggered events. Many bugs occur as a consequence of JavaScript's ability to manipulate the structure of the page while the page is being parsed. Moreover, such bugs are often of small depth [9,14], making web pages a suitable target for experiments.

While our model of asynchronous programs allows arbitrary reordering of independent (incomparable) events, the task of capturing all inter-dependencies between the events in real web pages presents a challenge. These events may be executions of scripts with complex internal control-flow and data dependencies, as well as effect on the global state. Once a schedule is reordered, new events might appear, and some events might never trigger, which adds another layer of complexity to the problem of systematic testing. To address these difficulties, we implemented our schedule generator on top of a tool called $R^4$ [9]. $R^4$ consists of a comprehensive infrastructure for executing web pages, automatic triggering of user events, and most importantly, approximate replay of reordered schedules.

As our test data, we randomly selected 24 websites of companies listed among the top 100 of Fortune 500 companies. The websites are listed in Table 1. For each website, we use $R^4$ to record a base schedule and construct the happens-before relation (the partial order). We then generate $d$-hitting families for $d = 2$ and $d = 3$. We use $R^4$ again to replay the generated schedules, and for each schedule we compare detected errors and exceptions, final HTML state, and visual differences with respect to the base schedule. This mechanism for detecting bugs is an adaptation of the mechanism used by $R^4$ itself.

The shape of the partial orders occuring in the web page setting is visible from the second and the third column of Table 1. A typical website has most of the events concentrated in a backbone of very large height $h$, i.e., with $h$ proportional to the total number of events $n$. As a result, for $d = 3$ we chose to

**Table 1.** Experimental results. For each website, we show the number of events in the initial execution, the height of the partial order (happens-before graph), the number of schedules generated for $d = 2$, the number of schedules generated for $d = 3$, and the number of schedules for $d = 3$ with pruning. In the last column, the checkmark denotes that all of the pruned schedules were successfully executed.

| Website | # Events | Height | $d = 2$ | $d = 3$ | $d = 3$ (pruned) | $d = 3$ (success) |
|---|---|---|---|---|---|---|
| abc.xyz | 337 | 288 | 2 | 561 | 0 | ✓ |
| newscorp.com | 1362 | 875 | 2 | 2689 | 100 | ✓ |
| thehartford.com | 2018 | 1547 | 2 | 3913 | 138 | |
| www.allstate.com | 4534 | 3822 | 2 | 9023 | 106 | |
| www.americanexpress.com | 2971 | 2586 | 2 | 5897 | 340 | ✓ |
| www.bankofamerica.com | 2305 | 2095 | 2 | 4561 | 150 | ✓ |
| www.bestbuy.com | 301 | 248 | 2 | 576 | 10 | ✓ |
| www.comcast.com | 188 | 118 | 2 | 337 | 16 | ✓ |
| www.conocophillips.com | 4184 | 3478 | 2 | 8286 | 248 | ✓ |
| www.costco.com | 7331 | 6390 | 2 | 14614 | 364 | |
| www.deere.com | 2286 | 1902 | 2 | 4516 | 236 | ✓ |
| www.generaldynamics.com | 2820 | 2010 | 2 | 5611 | 272 | |
| www.gm.com | 2337 | 1473 | 2 | 4600 | 94 | ✓ |
| www.gofurther.com | 1117 | 638 | 2 | 2154 | 568 | ✓ |
| www.homedepot.com | 3780 | 2100 | 2 | 7515 | 1526 | |
| www.humana.com | 5611 | 4325 | 2 | 11174 | 2058 | |
| www.johnsoncontrols.com | 2953 | 2395 | 2 | 5881 | 450 | |
| www.jpmorganchase.com | 4134 | 3519 | 2 | 8247 | 1316 | |
| www.libertymutual.com | 3885 | 3560 | 2 | 7735 | 324 | |
| www.lowes.com | 6938 | 4383 | 2 | 13778 | 3438 | |
| www.massmutual.com | 3882 | 3313 | 2 | 7682 | 1852 | |
| www.morganstanley.com | 2752 | 2301 | 2 | 5402 | 128 | |
| www.utc.com | 4081 | 3266 | 2 | 8100 | 206 | |
| www.valero.com | 2116 | 1849 | 2 | 4178 | 38 | ✓ |

implement the $O(n^{d-2}) = O(n)$ construction from subsection 3.2, since 3-hitting families from Sections 4 and 5 of size $O(h)$ and $O(h^2)$ might not have provided added benefit on our set of examples.

In fact, we were able to modify the $O(n^{d-2})$ construction in the following way. Since we are building on top of $R^4$, which in turn is built on top of a race detecting tool called EventRacer [14], we get information about races for free. A race consists of a pair of events accessing the same memory location, with at least one of them writing to this location. Here, the definition of memory locations is taken broadly to capture the DOM and the JavaScript memory [14,9]. Events that do not participate in races commute with all other events, so they need not be reordered. The information about races can be incorporated into the construction as follows. For $d = 3$, instead of selecting $a_1$ arbitrarily, we select it from the events that participate in races. We then perform the left-to-right and right-to-left traversals as usual. In total, the number of generated

```
<img src="..." onload="javascript:loaded()"/>

<script>
  function loaded() {
    document.getElementById('p').innerHTML = 'Loaded';
  }
</script>

<p id="p">Waiting...</p>
```

**Fig. 1.** Example of bugs of depth $d = 2$ and $d = 3$.

schedules is $2r$, where $r$ is the number of events participating in races. This number can be significantly smaller than $2n$, as can be seen in the fifth and sixth columns of Table 1. The information about races can be incorporated into other constructions as well: The events that do not need to be reordered can be merged together in a preprocessing step.

Even with the modification, some websites turned out to be too complex for our prototype. In particular, for some websites we failed to execute all of the generated schedules in the $d = 3$ case. The last column in Table 1 shows a checkmark for the websites with all schedules executed.

Typical examples of bugs of depths $d = 2$ and $d = 3$ are shown in Fig. 1. In the example, the image has an on-load event handler that calls the function `loaded()` once the image is loaded. The function, defined in a separate `<script>` block, changes the text of the paragraph `p` into *Loaded*. There are two potential bugs in this example. The first one is of depth $d = 2$, and it occurs if the image is loaded quickly (for example, from the cache), before the browser parses the `<script>` block. In this case, the on-load handler tries to call an undefined function. The second bug is of depth $d = 3$, and it occurs if the handler is executed after the `<script>` block is parsed, but before the `<p>` tag is parsed. In this case, the function `loaded()` tries to access a non-existent HTML element.

We detected several occurences of such bugs on the websites we have tested. The bugs manifest themselves as null-object JavaScript exceptions, together with visual differences—missing icons, buttons, and ads.

Unfortunately, the bug detection mechanism we use is fairly limited, as the HTML state or visual difference may be a result of dynamic data being loaded from an external source (for instance, a Twitter feed). In addition, unlike in the stateless model-checking approach used by $R^4$, where races are inverted one at a time, in our approach it is not easy to localize bugs. Better bug detection and localization is a topic for future work.

## 7 Conclusions

We have introduced $d$-hitting families as a common framework for systematic testing of asynchronous programs and studied the size of optimal $d$-hitting families for trees and related partial orders.

We have shown that a range of combinatorial techniques can be used to construct $d$-hitting families: We use a greedy approach, a randomized approach, and a construction based on DFS traversals; we also develop a direct inductive construction and a construction based on what we call patterns. The number of schedules in the pattern-based construction is polynomial in the height of the tree—for balanced trees, this is exponentially less than the total number of nodes.

Even with $d = 2$ we are able to detect bugs in real websites, and for $d = 3$ the size of 3-hitting families remains small. We observe that for $d \geq 3$ optimizing the size of the family is essential: even after pruning, the number of nodes in the trees, $n$, can be in the order of hundreds or even thousands. For $d = 3$, this makes the naive $O(n^d)$ construction practically infeasible, which suggests the use of more efficient families, say of size $O(n^{d-2})$ or $f(d)\,\mathrm{polylog}(n)$.

## References


1. Noga Alon and Joel H. Spencer. *The Probabilistic Method*. Wiley, 2008. 3rd edition.
2. Christoph Ambühl, Monaldo Mastrolilli, Nikolaus Mutsanas, and Ola Svensson. Precedence constraint scheduling and connections to dimension theory of partial orders. *Bulletin of the EATCS*, 95:37–58, 2008.
3. E. W. Miller Ben Dushnik. Partially ordered sets. *American Journal of Mathematics*, 63(3):600–610, 1941.
4. Sebastian Burckhardt, Pravesh Kothari, Madanlal Musuvathi, and Santosh Nagarakatte. A randomized scheduler with probabilistic guarantees of finding bugs. In *Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2010, Pittsburgh, Pennsylvania, USA, March 13-17, 2010*, pages 167–178, 2010.
5. Parinya Chalermsook, Bundit Laekhanukit, and Danupon Nanongkai. Graph products revisited: Tight approximation hardness of induced matching, poset dimension and more. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1557–1576. SIAM, 2013.
6. Michael Emmi, Shaz Qadeer, and Zvonimir Rakamaric. Delay-bounded scheduling. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 411–422, 2011.
7. Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar $\mathcal{F}$-deletion: Approximation, kernelization and optimal FPT algorithms. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 470–479. IEEE Computer Society, 2012.
8. Rajneesh Hegde and Kamal Jain. The hardness of approximating poset dimension. *Electronic Notes in Discrete Mathematics*, 29:435–443, 2007.
9. Casper Svenning Jensen, Anders Møller, Veselin Raychev, Dimitar Dimitrov, and Martin T. Vechev. Stateless model checking of event-driven applications. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2015,*

*part of SLASH 2015, Pittsburgh, PA, USA, October 25-30, 2015*, pages 57–73, 2015.

10. L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13(4):383–390, 1975.
11. Shan Lu, Soyeon Park, Eunsoo Seo, and Yuanyuan Zhou. Learning from mistakes: a comprehensive study on real world concurrency bug characteristics. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2008, Seattle, WA, USA, March 1-5, 2008*, pages 329–339, 2008.
12. Madan Musuvathi and Shaz Qadeer. CHESS: systematic stress testing of concurrent software. In *Logic-Based Program Synthesis and Transformation, 16th International Symposium, LOPSTR 2006, Venice, Italy, July 12-14, 2006, Revised Selected Papers*, pages 15–16, 2006.
13. Shaz Qadeer and Jakob Rehof. Context-bounded model checking of concurrent software. In *Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, pages 93–107, 2005.
14. Veselin Raychev, Martin T. Vechev, and Manu Sridharan. Effective race detection for event-driven programs. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications, OOPSLA 2013, part of SPLASH 2013, Indianapolis, IN, USA, October 26-31, 2013*, pages 151–166, 2013.
15. Alexander Sapozhenko. On the complexity of disjunctive normal forms obtained with a gradient algorithm. In *Diskretnyj Analiz (Discrete Analysis)*, volume 21, pages 62–71. Institute for Mathematics in the Siberian Section of the Academy of Sciences, Novosibirsk, 1972. In Russian.
16. Bernd S.W. Schröder. *Ordered Sets: An Introduction*. Springer, 2003.
17. S. K. Stein. Two combinatorial covering theorems. *J. Comb. Theory, Ser. A*, 16(3):391–397, 1974.
18. William T. Trotter. A generalization of Hiraguchi's: Inequality for posets. *J. Comb. Theory, Ser. A*, 20(1):114–123, 1976.
19. W.T. Trotter. *Combinatorics and Partially Ordered Sets: Dimension Theory*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 2001.
20. Mihalis Yannakakis. The complexity of the partial order dimension problem. *SIAM Journal on Algebraic Discrete Methods*, 3(3):351–358, 1982.

# A Antichains

## A.1 Calculation for the probabilistic method

The inequality in question is as follows:

$$n^d \cdot \left(1 - \frac{1}{d!}\right)^k < 1,$$

which can be rewritten as

$$k > \ln n \cdot d \cdot \frac{1}{-\ln(1 - 1/d!)}.$$

Recall that $-\ln(1-x) = x + \frac{x^2}{2} + \ldots \geq x$, so $-1/\ln(1-x) \leq 1/x$. Therefore, it suffices to take $k$ so that

$$k > \ln n \cdot d \cdot 1/(1/d!) = (d! \cdot d) \log n / \log e.$$

## A.2 Calculation for the greedy approach

We use the following formulation of the greedy covering lemma.

**Lemma 17.** *Suppose every element $e \in R = R_1 \cup \ldots R_s$ is contained in at least $\gamma s$ sets from $R_1, \ldots, R_s$, where $0 < \gamma \leq 1$. Then the size of any greedy cover does not exceed*

$$\left\lceil \frac{1}{\gamma} \ln^+(\gamma |R|) \right\rceil + \frac{1}{\gamma},$$

*where $\ln^+(x) = \max(0, \ln x)$ and $\ln x$ is the natural logarithm.*

Recall that our $|R| \leq n^d$ and $\gamma = 1/d!$. Observe that $n^d \geq d!$ since $n \geq d$, so $\ln^+(\gamma|R|) \leq \ln(n^d/d!)$. Therefore, the size of a greedy cover is at most

$$\begin{aligned} &\lceil d! \cdot \ln(n^d/d!) \rceil + d! \\ &\leq d!\, d \ln n - d! \ln(d!) + d! + 1 \\ &\leq d!\, d \ln n - \Theta(d!\, d \ln d). \end{aligned}$$

## A.3 Lower bound in the general case $d \geq 3$

Fix $n$ and $d \geq 3$. Denote $r = \lfloor (d-1)/2 \rfloor \geq 1$ and observe that $2r + 1 \leq d$. Take any $d$-hitting family $F = \{\alpha_1, \ldots, \alpha_k\}$ and consider the following matrix $B = (b_{ij})$ of size $k \times (n-1)_r$ where $(x)_r$ stands for $x(x-1)\ldots(x-r+1)$, the number of arrangements (the falling factorial). The columns of $B$ are indexed by all $r$-tuples of distinct elements from $\{v_1, \ldots, v_{n-1}\}$, of which there are exactly $(n-1)_r$. Let $(a_1, \ldots, a_r)$ be the $j$th such tuple, then the entry $b_{ij}$ is the number of elements from $\{a_1, \ldots, a_r\}$ that the schedule $\alpha_i$ places before $v_n$.

We claim that all columns of $B$ are pairwise distinct. Indeed, if for some $j' \neq j''$ and all $i$ it holds that $b_{ij'} = b_{ij''}$, then, for all $s \in \{0, \ldots, r\}$, no schedule from $F$ can place exactly $s$ elements from the $j'$th tuple before $v_n$ without also placing exactly $s$ elements from the $j''$th tuple before $v_n$, and vice versa. Since the $j'$th and $j''$th $r$-tuples —call them $\boldsymbol{a}'$ and $\boldsymbol{a}''$— are different, this implies that $F$ cannot be $d$-hitting. Indeed, in the case where $\boldsymbol{a}'$ and $\boldsymbol{a}''$ have no event in common, this is obvious: consider any $d$-tuple where all events from $\boldsymbol{a}'$ come before $v_n$ and all events from $\boldsymbol{a}''$ after $v_n$. But if $\boldsymbol{a}'$ and $\boldsymbol{a}''$ have, say, $\ell > 0$ events in common, then putting all the events of $\boldsymbol{a}'$ before $v_n$ and the remaining $r-\ell$ events of $\boldsymbol{a}''$ after $v_n$ produces a $d$-tuple that avoids getting hit by schedules from $F$ (note that $r > \ell$ as $\boldsymbol{a}'$ and $\boldsymbol{a}''$ are different).

Now, since each $b_{ij}$ can only assume values from the set $\{0, 1, \ldots, r\}$, it follows that $B$ cannot have more than $(r+1)^k$ columns. Therefore, $(n-1)_r \leq (r+1)^k$, and so $k \geq \log(n-1)_r / \log(r+1)$. Recall that $(x)_r = \binom{x}{r} \cdot r! \geq (x/r)^r \cdot r!$; we have

$$\begin{aligned} k &\geq \log\left(\left(\frac{n-1}{r}\right)^r \cdot r!\right) \Big/ \log(r+1) \\ &= \frac{r\log(n-1) - r\log r + \log r!}{\log(r+1)} \\ &= \frac{r}{\log(r+1)} \cdot \log(n-1) + w(r) \end{aligned}$$

where

$$w(r) = \frac{\log r! - r\log r}{\log(r+1)} \approx \frac{-r\log e + (\log r + \log \pi + 1)/2}{\log(r+1)}.$$

Substituting $r = \lfloor (d-1)/2 \rfloor$ gives the desired result, because

$$\begin{aligned} \frac{r}{\log(r+1)} &= \frac{\left\lfloor \frac{d-1}{2} \right\rfloor}{\log\left(\left\lfloor \frac{d-1}{2} \right\rfloor + 1\right)} \geq \frac{\frac{d-2}{2}}{\log\left(\frac{d-1}{2} + 1\right)} \\ &= \frac{d-2}{2\log\left(\frac{d+1}{2}\right)} = \frac{d-2}{2\log(d+1) - 2} \geq \frac{d}{2\log(d+1)} \end{aligned}$$

and $\log(n-1) = \log(n \cdot (1 - 1/n)) \geq \log n - 1$ for $n \geq d \geq 2$.

# B Trees and double trees, $d = 3$

## B.1 Proof of Lemma 9

We will prove the statement for $\left[\, A_h \middle| B_h \,\right]$; the proof for $\left[\, C_h \middle| D_h \,\right]$ is completely analogous. First consider the case when at least one of the events $a_1$ and $a_2$ belongs to the set $\{(-1, \varepsilon), (+1, \varepsilon)\}$. Taking into account symmetries of the setting, assume without loss of generality that $a_1 = (-1, \varepsilon)$. It suffices to show that all events of $\mathcal{D}^h$ except for $(-1, \varepsilon)$ appear at least once in the second halves of the schedules of $\left[\, A_h \middle| B_h \,\right]$, i.e., in the matrix $B_h$. This, however, easily follows from

the observation that the last two rows of $B_h$ mention (all events of) $\lambda(0)$ and $\lambda(1)$, as well as $(+1, \varepsilon)$. So in this case the statement of the lemma holds.

Now consider the case where both $a_1$ and $a_2$ come from the union of the sets $\mathcal{D}^{h-1}(0)$ and $\mathcal{D}^{h-1}(1)$. If they are both from the same set $\mathcal{D}^{h-1}(\ell)$ for some $\ell \in \{0, 1\}$, then the statement of the lemma follows from the inductive assumption, because $A_h$ contains $A_{h-1}(\ell)$ as a submatrix and, similarly, $B_h$ contains $B_{h-1}(\ell)$ as a submatrix. Otherwise $a_1$ and $a_2$ come from different sets; then one of them gets mentioned in $\lambda(0)$ and the other one in $\lambda(1)$, so the last two rows of the matrix $\big[\, A_h \big| B_h \,\big]$ satisfy the conditions of the lemma. This completes the proof.

### B.2 Proof of Theorem 7

Take an arbitrary triple $\boldsymbol{a} = (a_1, a_2, a_3)$ from $\mathcal{D}^{h+1}$; the statement of the theorem means that, whenever there is a schedule for $\mathcal{D}^{h+1}$ that hits $\boldsymbol{a}$, there is also a schedule in $M_{h+1}$ that hits $\boldsymbol{a}$.

Suppose $a_i = (s_i, x_i)$ where $x_i \in \{0, 1\}^*$ for $i = 1, 2, 3$. Let $p$ be the longest common prefix of $x_1$, $x_2$, and $x_3$. If $p \neq \varepsilon$, then we can recurse into $\mathcal{D}^h(\ell)$ where $\ell \in \{0, 1\}$ is the first symbol of $p$; this will preserve correctness of our arguments since every row of $M_h$ forms a subsequence of some row of $M_{h+1}$. We thus assume without loss of generality that $p = \varepsilon$. If one or two of $x_1$, $x_2$, and $x_3$ is/are $\varepsilon$, then the problem becomes easier, because the events $(-1, \varepsilon)$ and $(+1, \varepsilon)$ always occur first or last in a schedule; accordingly, we will henceforth assume that $x_i = \ell_i x'_i$ where $\ell_i \in \{0, 1\}$ and $x'_i \in \{0, 1\}^*$ for all $i$.

Note that the bit $\ell_i$ indicates whether $a_i$ comes from $\mathcal{D}^h(0)$ or $\mathcal{D}^h(1)$. Our assumption $p = \varepsilon$ means that one of the events $a_1$, $a_2$, and $a_3$ comes from one of these partial orders and the other two events from the other order. By symmetry, we will assume, again without loss of generality, that one event belongs to $\mathcal{D}^h(1)$ and two events to $\mathcal{D}^h(0)$. We split the argument into two cases according to which of $a_1$, $a_2$, and $a_3$ belongs to $\mathcal{D}^h(1)$.

In the first case, the only event coming from $\mathcal{D}^h(1)$ is $a_2$; it comes after $a_1 \in \mathcal{D}^h(0)$ but before $a_3 \in \mathcal{D}^h(0)$. We will use Lemma 9 for the double tree $\mathcal{D}^h(0)$ of half-height $h$ and for $\big[\, A_h \big| B_h \,\big]$: since the triple $\boldsymbol{a} = (a_1, a_2, a_3)$ respects $\mathcal{D}^{h+1}$, it follows that the pair $(a_1, a_3)$ respects $\mathcal{D}^h(0)$, and thus there exists a schedule for $\mathcal{D}^h(0)$ where $a_1$ comes before $a_3$. By Lemma 9, the matrix $M' = \big[\, A_h(0) \big| B_h(0) \,\big]$ contains such a schedule where additionally $a_1$ appears in the first half and $a_3$ in the second half, i.e., they appear in $A_h(0)$ and $B_h(0)$, respectively. But the matrix $M'$ is a submatrix of $M_{h+1}$ and, what's crucial, between the left and right halves of $M'$ in $M_{h+1}$ comes the matrix $M'' = \big[\, A_h(1) \big| B_h(1) \,\big]$, each row of which is a complete schedule for the order $\mathcal{D}^h(1)$. Naturally, the event $a_2$ appears in every row of $M''$. To sum up, the matrix $M_{h+1}$ has a row where $a_1$ appears first (in the $A_h(0)$ block), $a_2$ second (in $A_h(1)$ or $B_h(1)$), and $a_3$ third (in $B_h(0)$). This completes our analysis of the first case.

Now consider the case when either $a_1$ or $a_3$ belongs to $\mathcal{D}^h(1)$, and the other two events belong to $\mathcal{D}^h(0)$. In this case, one of the four rows of $M_{h+1}$ that mention $\lambda(0)$, $\lambda(1)$, $\rho(0)$, and $\rho(1)$ defines an appropriate schedule. For example,

if $a_3 \in \mathcal{D}^h(1)$ and $a_1, a_2 \in \mathcal{D}^h(0)$, then either $\lambda(0)$ or $\rho(0)$ puts $a_1$ before $a_2$, and then $a_3$ appears in both $\lambda(1)$ and $\rho(1)$. The other subcase, $a_1 \in \mathcal{D}^h(1)$, is analogous. This completes our analysis of the second case and the proof of Theorem 7.

# C Trees and aribtrary $d \geq 3$

## C.1 Proof of Claim 14

Consider the tree $T$ as a partial order, $(T, \leq)$, where the root is the smallest element. Let $X \subseteq T$. It is immediate that the restriction of $\leq$ to $[X]$ is also a tree, $([X], \leq)$. Suppose $[X] = L \cup B \cup U$ where $L$ is the set of leaves of this new tree $([X], \leq)$, $B$ is the set of its non-leaf nodes with more than 1 child, and $U$ is the set of its non-leaf nodes with exactly 1 child. Sets $L$, $B$, and $U$ are disjoint.

We now trace the "provenance" of elements of these sets, i.e., look into why they are included in $[X]$. It is clear that $L \subseteq X$ and $U \subseteq X$, because only nodes with 2 or more children can belong to $[X] \setminus X$. Nodes of the set $B$ are the only "branching points" of the tree $([X], \leq)$, and thus their number cannot exceed $|L| - 1$. More formally, denote by $n_i$ the number of nodes of $([X], \leq)$ with exactly $i$ children, $i \geq 0$. As each edge in the graph departs from some node and arrives at some node,

$$\sum_{v \in [X]} \text{indeg}(v) = \sum_{v \in [X]} \text{outdeg}(v).$$

The left-hand side of this equation is $n - 1$, where $n = |[X]|$, because each node except for the root has a parent. Therefore,

$$\begin{aligned} n - 1 &= \sum_{i \geq 0} n_i \cdot i, \\ n_0 + n_1 + n_2 + \ldots - 1 &= n_1 + 2n_2 + 3n_3 + \ldots, \\ n_0 + n_1 + n_2 + \ldots - 1 &\geq n_1 + 2n_2 + 2n_3 + \ldots \end{aligned}$$

Denote $r = |B| = n_2 + n_3 + \ldots$, then $n_0 + n_1 + r - 1 \geq n_1 + 2r$, and so $r \leq n_0 - 1$, which is the same as $|B| \leq |L| - 1$.

To sum up, $|[X]| = |L \cup U| + |B| \leq |L \cup U| + |L| - 1$. Since $L \cup U \subseteq X$ as argued above, we conclude that $|[X]| \leq 2|X| - 1$.

## C.2 Proof of Lemmas 11 and 13

*Proof (of Lemma 11).* Recall that we need to show that for each admissible $d$-tuple $\boldsymbol{a}$ there exists a pattern $p$ such that $\boldsymbol{a}$ conforms to $p$. Take any such tuple $\boldsymbol{a} = (a_1, \ldots, a_d)$; since it is admissible, there exists a schedule $\alpha$ for $\mathcal{T}^h$ that hits $\boldsymbol{a}$. Consider the set $\{a_1, \ldots, a_d\}$ and take its lca-closure in $\mathcal{T}^h$: $D = [\{a_1, \ldots, a_d\}]$. Let $\preccurlyeq$ be the restriction of $\leq$ to $D$. Now for each non-leaf node

$v \in D$ in the partial order $(D, \preccurlyeq)$ define $\ell(v) = |v|$; again, recall that elements of $\mathcal{T}^h$ are binary strings from $\{0,1\}^{\leq h}$. Furthermore, consider each node $v \in D$ in $(D, \preccurlyeq)$ with outdegree 2; if $v'$ and $v''$ are the children of $v$ in $(D, \preccurlyeq)$, then $v'$ and $v''$ lie in different principal subtrees of $v$ in $\mathcal{T}^h$ (because otherwise the equality $\mathrm{lca}(v', v'') = v$ cannot hold); that is, $v' = v \cdot 0 \cdot u'$ and $v'' = v \cdot 1 \cdot u''$ for some strings $u', u'' \in \{0,1\}^*$. Accordingly, define $s(v') = 0$ and $s(v'') = 1$. Finally, take the schedule $\alpha$ and restrict it to the set $D$; denote the obtained schedule by $\pi$.

It is not difficult to check that the tuple $\boldsymbol{a}$ conforms to the constructed pattern $p = (D, \preccurlyeq, s, \ell, \pi)$. Note that the upper bound on $|D|$ is by Claim 14 and the upper bound on the number of non-leaf nodes in $(D, \preccurlyeq)$ holds by the following argument. Let $m \leq d$ be the number of leaves of $(D, \preccurlyeq)$ in the set $\{a_1, \ldots, a_d\}$; then $(D, \preccurlyeq)$ has exactly $m-1$ binary nodes (none of them leaves). Furthermore, all non-leaf unary nodes in $(D, \preccurlyeq)$ cannot belong to the difference $D \setminus \{a_1, \ldots, a_d\}$ and thus all lie in the set $\{a_1, \ldots, a_d\}$; their number cannot exceed the number of all non-leaf nodes in $\{a_1, \ldots, a_d\}$, i.e., is at most $d-m$. Hence, the total number of non-leaf nodes in $(D, \preccurlyeq)$ does not exceed $(m-1) + (d-m) = d-1$. This concludes the proof. □

*Proof (of Lemma 13).* We need to count the number of patterns, up to isomorphism. A pattern is fully specified by its components:

— the binary tree $(D, \preccurlyeq)$ with at most $2d-1$ nodes and the partial function $s$ that specifies a planar embedding of this tree—the total number of such embeddings (for all trees) is at most $4^{2d-1}/3$;
— the partial function $\ell$ with domain of size at most the number of non-leaf nodes in $D$ (i.e., at most $d-1$), and co-domain of size $h$—the number of suitable functions is at most $h^{d-1}$;
— the schedule $\pi$ for $(D, \preccurlyeq)$—of which there are at most $(2d-1)!$.

Thus the total number of patterns does not exceed

$$4^{2d-1}/3 \cdot h^{d-1} \cdot (2d-1)! = \exp(d) \cdot h^{d-1}.$$

This completes the proof. □

### C.3 Proof of Lemma 12

Fix any pattern $p = (D, \preccurlyeq, s, \ell, \pi)$. Recall that we need to find a schedule $\alpha_p$ that hits all $d$-tuples $\boldsymbol{a} = (a_1, \ldots, a_d)$ conforming to $p$. We will pursue the following strategy. We will cut the tree $\mathcal{T}^h$ into multiple pieces; this cutting will be entirely determined by the pattern $p$, independent of any individual $\boldsymbol{a}$. Each piece in the cutting will be associated with some element $c \in D$, so that each element of $D$ can have several pieces associated with it. In fact, every piece will form a subtree of $\mathcal{T}^h$ (although this will be of little importance). The key property is that, for every $d$-tuple $\boldsymbol{a} = (a_1, \ldots, a_d)$ conforming to $p$, if $i$ is the isomorphism from Definition 16, then each event $a_k$, $1 \leq k \leq d$, will belong to a piece associated

with $i^{-1}(a_k)$. As a result, the desired schedule $\alpha_p$ can be obtained in the following way: arrange the pieces according to how $\pi$ schedules elements of $D$ and pick any possible schedule inside each piece. This schedule will be guaranteed to meet the requirements of the lemma.

We now show how to implement this strategy. We describe a procedure that, given $p$, constructs a suitable $\alpha_p$. To simplify the presentation, we will describe cutting of $\mathcal{T}^h$ and constructing $\alpha_p$ simultaneously, although they can be performed separately. The cutting itself is defined by the following formalism. For each element $c \in D$, we define a set $E(c) \subseteq \mathcal{T}^h$, with the intention that events from $E(c)$ point to the roots of all pieces associated with $c$. The pieces themselves stretch out down the tree up to (and including) layer $\ell(c)$; as the value $\ell(c)$ is undefined for leaves of $(D, \preccurlyeq)$, we will instead use the extension of $\ell$ that assigns $\ell(c) = h$ for all leaves $c$ of $(D, \preccurlyeq)$, abusing the notation $\ell$. As we go along, we add more and more events to the schedule $\alpha_p$, constructing it on the way; we will refer to this as *scheduling* these events. The events in $E(c)$ can be thought of as *enabled* after scheduling the events from previously considered pieces: that is, all these events have not been scheduled yet, but all their immediate predecessors (parents) in $(D, \preccurlyeq)$ have. This will allow us to schedule the pieces rooted at $E(c)$ at any suitable moment. We will not give any "prior" definition of $E(c)$: these sets will only be determined during the process.

Overall, the *invariant* of the procedure is that, when we define $E(c)$, the events in $E(c)$ form an antichain, are enabled (not scheduled yet, but all predecessors already scheduled), and belong to layers $\leq \ell(c)$ of the partial order $\mathcal{T}^h$.

Let us now fill in the missing details of the process. At first, no events are scheduled, and the set $E(c_*)$, where $c_*$ is the root of $(D, \preccurlyeq)$, is defined as the singleton $\{\varepsilon\}$; recall that $\varepsilon$ is the root of the tree $\mathcal{T}^h$. The procedure goes over the schedule $\pi$, which is part of the pattern $p$, and handles elements $c$ scheduled by $\pi$ one by one. The first element is, of course, the root of $(D, \preccurlyeq)$, which we called $c_*$. Note that at the beginning of the procedure, the invariant is satisfied.

To handle an element $c$ scheduled by $\pi$, our procedures performs the following steps. It first schedules all events in the set

$$U(c) = \{y \in \mathcal{T}^h \mid x \leq y \text{ for some } x \in E(c) \text{ and } |y| \leq \ell(c)\},$$

i.e., all events $x \in E(c)$ and all events that are successors of $x \in E(c)$ in layers up to and including $\ell(c)$. Note that this set $U(c)$ consists of a number of disjoint subtrees of the tree $\mathcal{T}^h$; these subtrees are the pieces that we previously discussed, and $U(c)$ is their union. Each piece is non-empty: for all $x \in E(c)$, the set of all $y$ such that $x \leq y$ and $|y| \leq \ell(c)$ contains at least the element $x$ itself, because, by our invariant, $\ell(x) \leq \ell(c)$; therefore, $E(c) \subseteq U(c)$. The pieces (subtrees) are disjoint because the events in $E(c)$ form an antichain. Finally, scheduling these pieces is possible because, on one hand, no $x \in E(c)$ has been scheduled previously and, on the other hand, all predecessors of $x \in E(c)$ have already been scheduled. Note that we can schedule all events from $U(c)$ in any order admitted by $\mathcal{T}^h$, for instance using lexicographic depth-first traversal.

After this, the procedure forms new sets $E$; the precise choice depends on the outdegree of $c$ in $(D, \preccurlyeq)$. Recall that this outdegree does not exceed 2 by our definition of the pattern. Observe that after scheduling the pieces associated with $c$, as described in the previous paragraph, the following events, for all $x \in E(c)$, are made enabled: $z \in \mathcal{T}^h \cap (\{0,1\}^{\ell(c)}.\{0,1\})$ with $x \leq z$. In fact, this set is empty iff $\ell(c) = h$; by our choice of $\ell$, this happens if and only if $d = 0$, i.e., when $c$ is a leaf of $(D, \preccurlyeq)$. In such a case, no new set $E$ is formed and the procedure proceeds to the next element of $\pi$. Otherwise $d \in \{1, 2\}$; we consider each case separately. If $d = 1$, then the element $c \in D$ has a single child in the tree $(D, \preccurlyeq)$. Denote this child by $c'$ and define

$$E(c') = \{z \in \{0,1\}^{\ell(c)+1} \mid x \leq z \text{ for some } x \in E(c)\}.$$

If $d = 2$, then the element $c \in D$ has two children in the tree $(D, \preccurlyeq)$. Let these children be $c_0$ and $c_1$, such that $s(c_r) = r$ for both $r \in \{0, 1\}$. We now split the set of newly enabled events as follows:

$$E(c_0) = \{\bar{z}.0 \in \{0,1\}^{\ell(c)+1} \mid x \leq \bar{z}.0 \text{ for some } x \in E(c)\},$$
$$E(c_1) = \{\bar{z}.1 \in \{0,1\}^{\ell(c)+1} \mid x \leq \bar{z}.1 \text{ for some } x \in E(c)\}.$$

Note that the events $z$ in $E(c')$ (or in $E(c_0)$ and $E(c_1)$, depending on $d$) form an antichain, are enabled, and, moreover, satisfy the inequality $\ell(z) \leq \ell(c')$, because $\ell(z) = \ell(c) + 1$ and $\ell(c) < \ell(c')$ by the choice of $\ell$. This ensures that during the run of the procedure the invariant is maintained.

It is not difficult to see that the described procedure outputs some schedule $\alpha_p$ for $\mathcal{T}^h$. We now show why this $\alpha_p$ satisfies our requirements. Indeed, pick any admissible $d$-tuple $\boldsymbol{a} = (a_1, \ldots, a_d)$ conforming to the pattern $p$; we need to check that $\alpha_p$ hits $\boldsymbol{a}$. In fact, by the choice of our strategy, it is sufficient to check that each event $a_k$, $1 \leq k \leq d$, belongs to a piece associated with the event $i^{-1}(a_k)$ where $i$ is the isomorphism from the definition of conformance. In other words, we need to ensure that each event $a_k$ belongs to the set $U(c)$ for $c = i^{-1}(a_k)$; we will prove a stronger claim that $a_k \in U(c) \cap \{0,1\}^{\ell(c)}$ for this $c$. Note that the choice of $U(c)$ is such that $U(c) \subseteq \{0,1\}^{\leq \ell(c)}$.

The proof of this claim follows our construction of $\alpha_p$. Indeed, consider the event $a_1$ first; we necessarily have $i^{-1}(a_1) = c_*$. By our definition of conformance, $a_1$ is on the $\ell(c_*)$th layer in the tree $\mathcal{T}^h$, that is, $|a_1| = \ell(c_*)$. By the description of our procedure, all events from $\{0,1\}^{\ell(c_*)}$ are associated with $c_*$, i.e., belong to $U(c_*)$ and and are thus scheduled during the first step of the procedure. Note that since $\ell(c) > \ell(c_*)$ for all $c \neq c_*$ in $D$ and $\ell$ correctly specifies the height in $\mathcal{T}^h$, none of the events $a_2, \ldots, a_d$ can be scheduled before $a_1$. Also observe that the existence of the isomorphism $i$ ensures that all the events $a_2, \ldots, a_d$ are successors of $a_1$.

It now remains to follow the inductive step: suppose the claim holds for an event $a_k$ with $i^{-1}(a_k) = c$ for some $c \in D$. As soon as our procedure schedules $U(c) \cap \{0,1\}^{\ell(c)}$, all its successors become enabled, because $U(c) \subseteq \{0,1\}^{\leq \ell(c)}$. We now need to consider three cases depending on the value of $d$. If $d = 0$,

there is nothing to prove. If $d = 1$, both successors of $a_k$ in $\mathcal{T}^h$ are included into $E(c') \subseteq U(c')$. where $c'$ is the only child of $c$ in $(D, \preccurlyeq)$. by our choice of $\ell$ it holds that $\ell(c') = |i(c')|$. Since the event $i(c')$ is a (not necessarily direct) successor of $i(c)$ and is different from $i(c)$, it follows that $x \leq i(c')$ for some event $x \in E(c)$. But then it follows that $i(c') \in U(c')$ by the choice of $U$. Similarly, consider $d = 2$. All 0-children and 1-children of $a_k$ in $\mathcal{T}^h$ are included in $E(c_0)$ and $E(c_1)$, respectively, where by $c_r$, $r \in \{0, 1\}$, we denote the (unique) child of $c$ in $(D, \preccurlyeq)$ that has $s(c_r) = r$. Since $s$ correctly specifies 0- and 1-principal subtree relations in $\mathcal{T}^h$, it follows that $i(c_r)$ belongs to the $r$-principal subtree of $i(c)$, for each $r \in \{0, 1\}$. So our choice of $E(c_0)$ and $E(c_1)$ ensures that, for each $r \in \{0, 1\}$, there exists an $x \in E(c_r)$ such that $x \leq i(c_r)$. The conditions on the layer are checked in the same way as in the case $d = 1$; the upshot is that $i(c_r) \in U(c_r) \cap \{0, 1\}^{\ell(c_r)}$ for both $r$. This completes the proof of the claim, from which the correctness of the procedure constructing $\alpha_p$ follows.

This concludes the proof of Lemma 12.

**Remark 18.** The proof above cuts the tree right below the layers specified by the function $\ell$; this choice is somewhat arbitrary and can be changed. Moreover, for presentational purposes we also decided to schedule all elements of sets $U(c)$ at once. This choice is essentially employing a *breadth-first* strategy: as soon as we get to process $c$, we necessarily schedule all possible candidates for its image $i(c)$. However, a *depth-first* strategy also works: in this strategy, elements $x \in E(c)$ are processed one-by-one. More precisely, the procedure can first schedule all elements of $U(c)$ that are successors of $x$, essentially going into the subtree of $\mathcal{T}^h$ rooted at $x$. After this, instead of switching to a different $x' \in E(c)$, the procedure could stay inside this subtree and follow, as usual, the guidance of $\pi$, *assuming* that the chosen subtree indeed contains $i(c)$. Only after scheduling *all* elements of the subtree (i.e., all $u \in \mathcal{T}^h$ such that $x \leq u$) does the procedure comes back to its set $E(c)$ and proceeds to the next candidate $x' \in E(c)$. In fact, during the run of this modified procedure *many* different sets $E(c)$ will be defined (as long as $c \neq c_*$); all these sets will be disjoint, and their union will be equal to the original set $E(c)$ as defined in the proof above.